16 | 最近邻检索（下）：如何用乘积量化实现“拍照识花”功能？

2020-05-01 陈东

讲述：陈东
时长 22:27 大小 20.56M

你好，我是陈东。

随着 AI 技术的快速发展，以图搜图、拍图识物已经是许多平台上的常见功能了。比如说，在搜索引擎中，我们可以直接上传图片进行反向搜索。在购物平台中，我们可以直接拍照进行商品搜索。包括在一些其他的应用中，我们还能拍照识别植物品种等等。这些功能都依赖于高效的图片检索技术，那它究竟是怎么实现的呢？今天，我们就来聊一聊这个问题。

聚类算法和局部敏感哈希的区别？

检索图片和检索文章一样，我们首先需要用向量空间模型将图片表示出来，也就是将一个图片对象转化为高维空间中的一个点。这样图片检索问题就又变成了我们熟悉的高维空间的相

似检索问题。

如果我们把每个图片中的像素点看作一个维度，把像素点的 RGB 值作为该维度上的值，那一张图片的维度会是百万级别的。这么高的维度，检索起来会非常复杂，我们该怎么处理呢？我们可以像提取文章关键词一样，对图片进行特征提取来压缩维度。

要想实现图片特征提取，我们有很多种深度学习的方法可以选择。比如，使用卷积神经网络（CNN）提取图片特征。这样，用一个 512 到 1024 维度的向量空间模型，我们就可以很好地描述图像了，但这依然是一个非常高的维度空间。因此，我们仍然需要使用一些近似最邻近检索技术来加速检索过程。

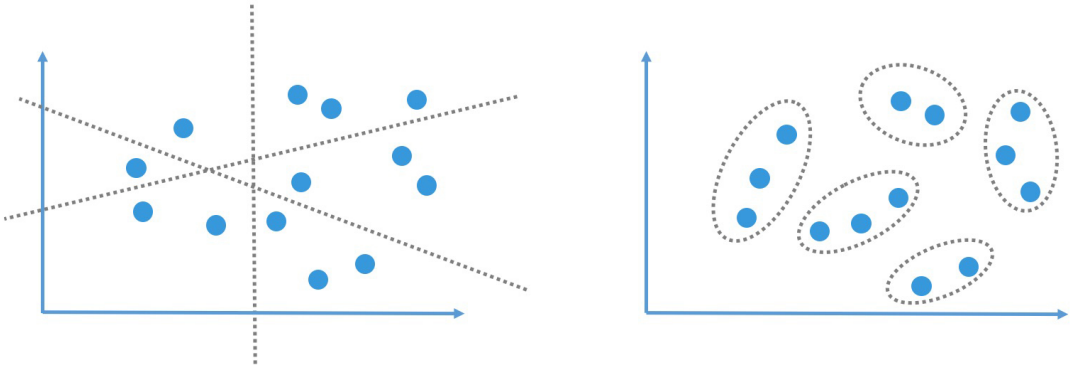
一种常用的近似最邻近检索方法，是使用局部敏感哈希对高维数据进行降维处理，将高维空间的点划到有限的区域中。这样，通过判断要查询的点所在的区域，我们就能快速取出这个区域的所有候选集了。

不过，在上一讲中我们也提到，局部敏感哈希由于哈希函数构造相对比较简单，往往更适合计算字面上的相似性（表面特征的相似性），而不是语义上的相似性（本质上的相似性）。这怎么理解呢？举个例子，即便是面对同一种花，不同的人在不同的地点拍出来的照片，在角度、背景、花的形状上也会有比较大的差异。也就是说，这两张图片的表面特征其实差异很大，这让我们没办法利用局部敏感哈希，来合理评估它们的相似度。

而且，局部敏感哈希其实是一种粒度很粗的非精准检索方案。以 SimHash 为例，它能将上百万的高维空间压缩到 64 位的比特位中，这自然也会损失不少的精确性。

表面特征差异很大的同一种花的对比示意图

因此，更常见的一种方案，是使用聚类算法来划分空间。和简单的局部敏感哈希算法相比，聚类算法能将空间中的点更灵活地划分成多个类，并且保留了向量的高维度，使得我们可以更准确地计算向量间的距离。好的聚类算法要保证类内的点足够接近，不同类之间的距离足够大。一种常见的聚类算法是 K-Means 算法（K- 平均算法）。



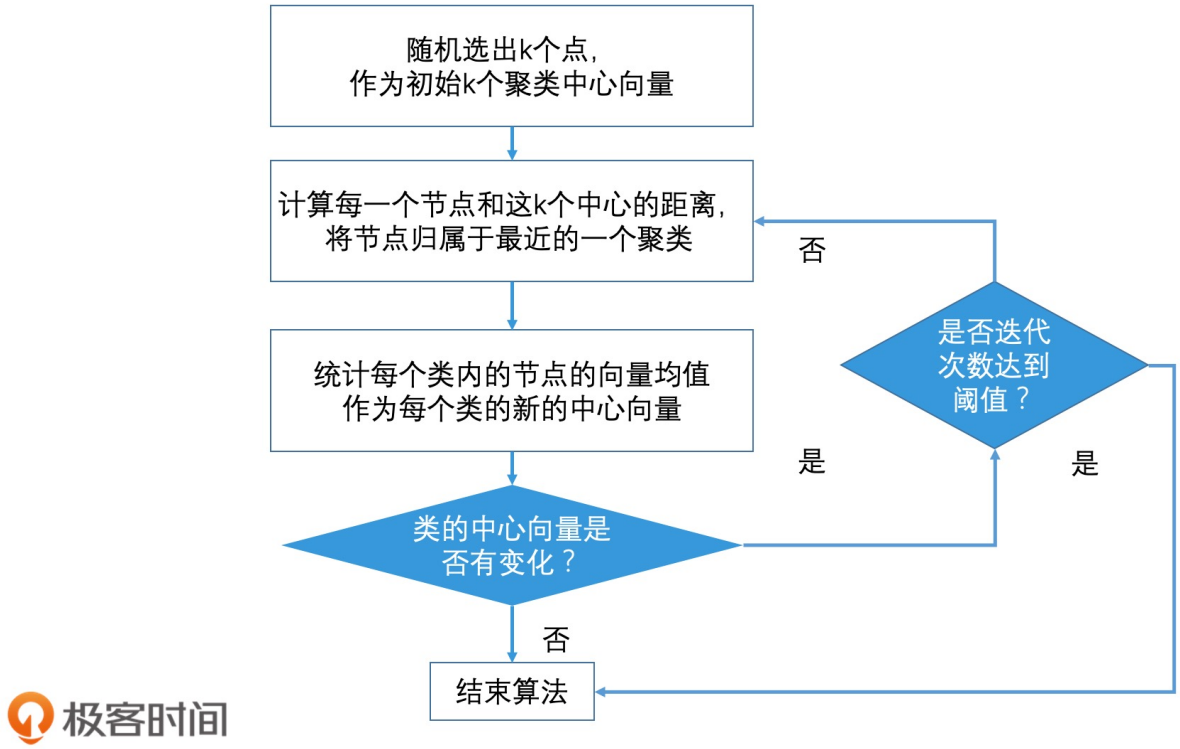
局部敏感哈希的空间划分　　聚类的空间划分

极客时间

局部敏感哈希空间划分 VS 聚类空间划分

K-Means 聚类算法的思想其实很“朴素”，它将所有的点划分为 k 个类，每个类都有一个**类中心向量**。在构建聚类的时候，我们希望每个类内的点都是紧密靠近类中心的。用严谨的数学语言来说，K-Means 聚类算法的优化目标是，**类内的点到类中心的距离均值总和最短**。因此，K-Means 聚类算法具体的计算步骤如下:

1. 随机选择 k 个节点，作为初始的 k 个聚类的中心；
2. 针对所有的节点，计算它们和 k 个聚类中心的距离，将节点归入离它最近的类中；
3. 针对 k 个类，统计每个类内节点的向量均值，作为每个类的新的中心向量；
4. 重复第 2 步和第 3 步，**重新计算每个节点和新的类中心的距离，将节点再次划分到最近的类中，然后再更新类的中心节点向量**。经过多次迭代，直到节点分类不再变化，或者迭代次数达到上限，我们停止算法。



K-Means 算法计算流程图

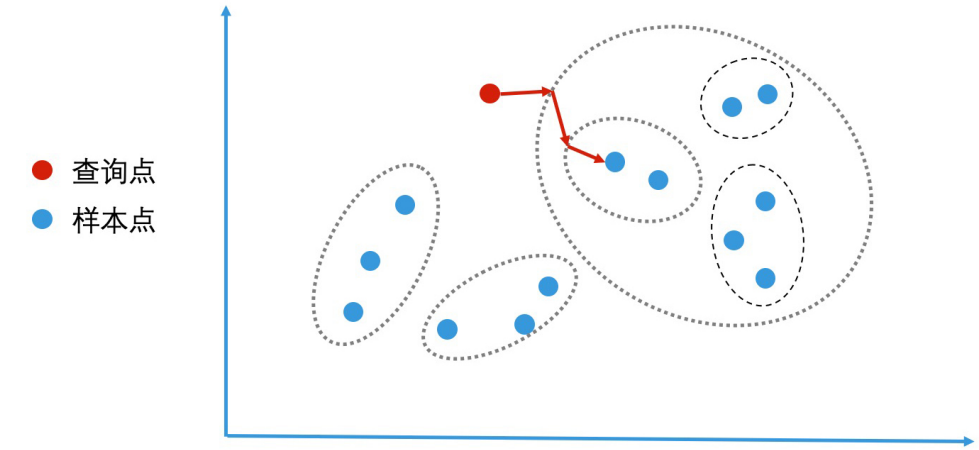
以上，就是 K-Means 聚类算法的计算过程了，那使用聚类算法代替了局部敏感哈希以后，我们该怎么进行相似检索呢？

如何使用聚类算法进行相似检索？

首先，对于所有的数据，我们先用聚类算法将它们划分到不同的类中。在具体操作之前，我们会给聚类的个数设定一个目标。假设聚类的个数是 1024 个，那所有的点就会被分到这 1024 个类中。这样，我们就可以用每个聚类的 ID 作为 Key，来建立倒排索引了。

建立好索引之后，当要查询一个点邻近的点时，我们直接计算该点和所有聚类中心的距离，将离查询点最近的聚类作为该点所属的聚类。因此，以该聚类的 ID 为 Key 去倒排索引中查询，我们就可以取出所有该聚类中的节点列表了。然后，我们遍历整个节点列表，计算每个点和查询点的距离，取出 Top K 个结果进行返回。

这个过程中会有两种常见情况出现。第一种，最近的聚类中的节点数非常多。这个时候，我们就计算该聚类中的所有节点和查询点的距离，这个代价会很大。这该怎么优化呢？这时，我们可以参考二分查找算法不断划分子空间划分的思路，使用层次聚类将一个聚类中的节点，再次划分成多个聚类。这样，在该聚类中查找相近的点时，我们通过继续判断查询点和哪个子聚类更相近，就能快速减少检索空间，从而提升检索效率了。



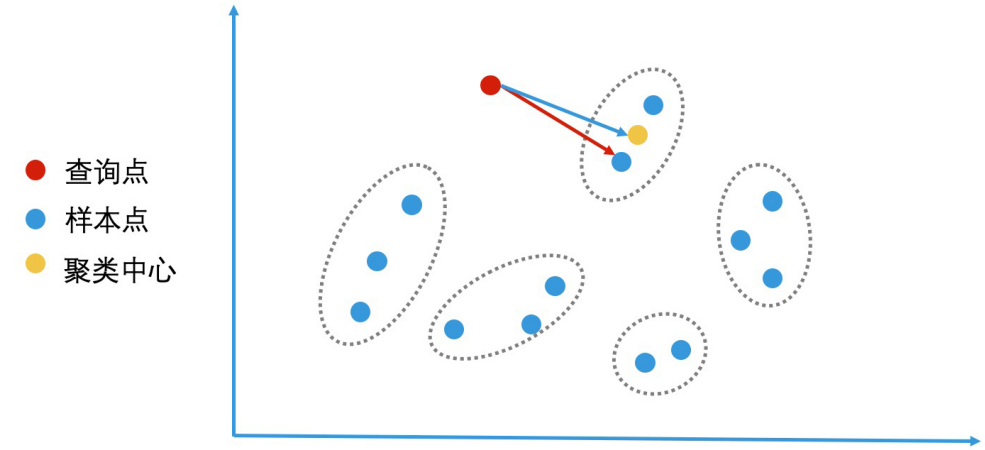
层次聚类检索过程示意图

第二种，该聚类中的候选集不足 Top K 个，或者我们担心聚类算法的相似判断不够精准，导致最近的聚类中的结果不够好。那我们还可以再去查询次邻近的聚类，将这些聚类中的候选集取出，计算每个点和查询点的距离，补全最近的 Top K 个点。

如何使用乘积量化压缩向量?

对于向量的相似检索，除了检索算法本身以外，如何优化存储空间也是我们必须要关注的一个技术问题。以 1024 维的向量为例，因为每个向量维度值是一个浮点数（浮点数就是小数，一个浮点数有 4 个字节），所以一个向量就有 4K 个字节。如果是上亿级别的数据，光是存储向量就需要几百 G 的内存，这会导致向量检索难以在内存中完成检索。

因此，为了能更好地将向量加载到内存中，我们需要压缩向量的表示。比如说，我们可以用聚类中心的向量代替聚类中的每个向量。这样，一个类内的点都可以用这个类的 ID 来代替和存储，我们也就节省了存储每个向量的空间开销。那计算查询向量和原始样本向量距离的过程，也就可以改为计算查询向量和对应聚类中心向量的距离了。



极客时间

用聚类中心代替样本点

想要压缩向量，我们往往会使用**向量量化**（Vector Quantization）技术。其中，我们最常用的是**乘积量化**（Product Quantization）技术。

乍一看，你会觉得乘积量化是个非常晦涩难懂的概念，但它其实并没有那么复杂。接下来，我就把它拆分成乘积和量化这两个概念，来为你详细解释一下。

量化指的就是将一个空间划分为多个区域，然后为每个区域编码标识。比如说，一个二维空间 <x,y> 可以被划为两块，那我们只需要 1 个比特位就能分别为这两个区域编码了，它们的空间编码分别是 0 和 1。那对二维空间中的任意一个点来说，它要么属于区域 0，要么属于区域 1。

这样，我们就可以用 1 个比特位的 0 或 1 编码，来代替任意一个点的二维空间坐标 <x,y> 了 。假设 x 和 y 是两个浮点数，各 4 个字节，那它们一共是 8 个字节。如果我们将 8 个字节的坐标用 1 个比特位来表示，就能达到压缩存储空间的目的了。前面我们说的用聚类 ID 代替具体的向量来进行压缩，也是同样的原理。

而**乘积指的是高维空间可以看作是由多个低维空间相乘得到的。**我们还是以二维空间 <x,y> 为例，它就是由两个一维空间和相乘得到。类似的还有，三维空间 <x,y,z> 是由一个二维空间 <x,y> 和一个一维空间相乘得到，依此类推。

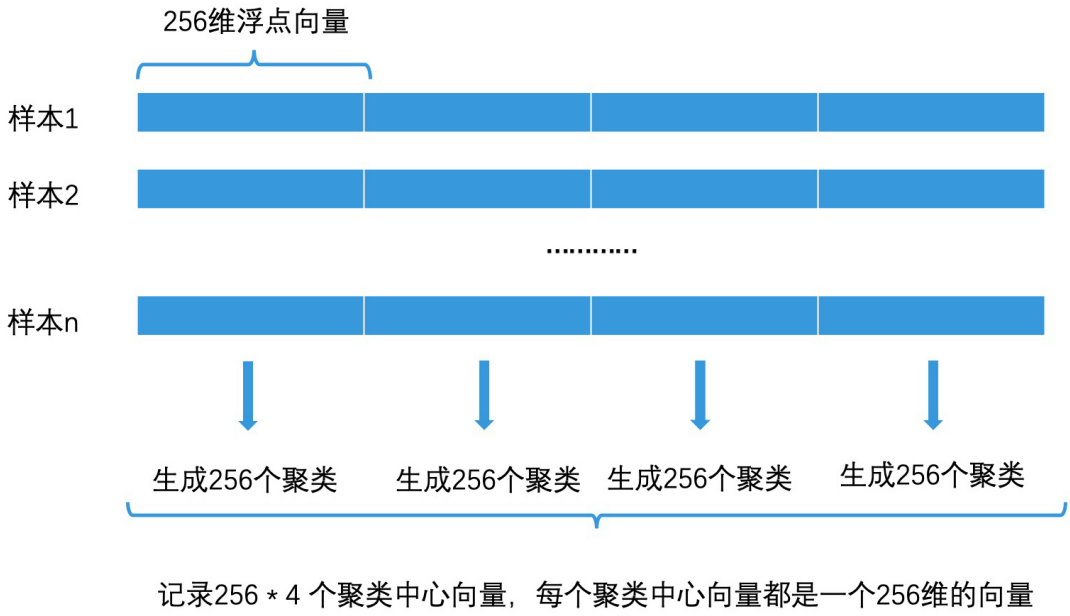
那将高维空间分解成多个低维空间的乘积有什么好处呢？它能降低数据的存储量。比如说，二维空间是由一维的 x 轴和 y 轴相乘得到。x 轴上有 4 个点 x1 到 x4，y 轴上有 4 个点 y1

到 y4，这四个点的交叉乘积，会在二维空间形成 16 个点。但是，如果我们仅存储一维空间中，x 轴和 y 轴的各 4 个点，一共只需要存储 8 个一维的点，这会比存储 16 个二维的点更节省空间。

总结来说，对向量进行乘积量化，其实就是将向量的高维空间看成是多个子空间的乘积，然后针对每个子空间，再用聚类技术分成多个区域。最后，给每个区域生成一个唯一编码，也就是聚类 ID。

好了，乘积量化压缩向量的原理我们已经知道了。接下来，我们就通过一个例子来说说，乘积量化压缩样本向量的具体操作过程。

如果我们的样本向量都是 1024 维的浮点数向量，那我们可以将它分为 4 段，这样每一段就都是一个 256 维的浮点向量。然后，在每一段的 256 维的空间里，我们用聚类算法将这 256 维空间再划分为 256 个聚类。接着，我们可以用 1 至 256 作为 ID，来为这 256 个聚类中心编号。这样，我们就得到了 256 * 4 共 1024 个聚类中心，每个聚类中心都是一个 256 维的浮点数向量（256 * 4 字节 = 1024 字节）。最后，我们将这 1024 个聚类中心向量都存储下来。



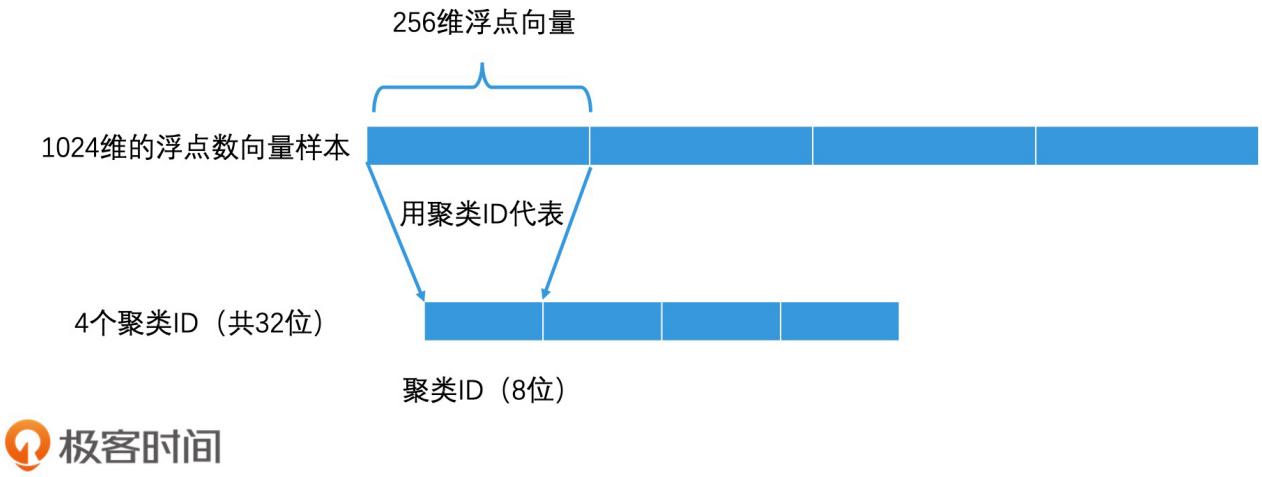
极客时间

记录256*4个聚类向量中心示意图

这样，对于这个空间中的每个向量，我们就不需要再精确记录它在每一维上的权重了。我们只需要将每个向量都分为四段，让**每段子向量都根据聚类算法找到所属的聚类，然后用它所**

属聚类的 ID 来表示这段子向量就可以了。

因为聚类 ID 是从 1 到 256 的，所以我们只需要 8 个比特位就可以表示这个聚类 ID 了。由于完整的样本向量有四段，因此我们用 4 个聚类 ID 就可以表示一个完整的样本向量了，也就一共只需要 32 个比特位。因此，一个 1024 维的原始浮点数向量（共 1024 * 4 字节）使用乘积量化压缩后，存储空间变为了 32 个比特位，空间使用只有原来的 1/1024。存储空间被大幅降低之后，所有的样本向量就有可能都被加载到内存中了。



压缩前后向量的存储空间对比图

如何计算查询向量和压缩样本向量的距离（相似性）？

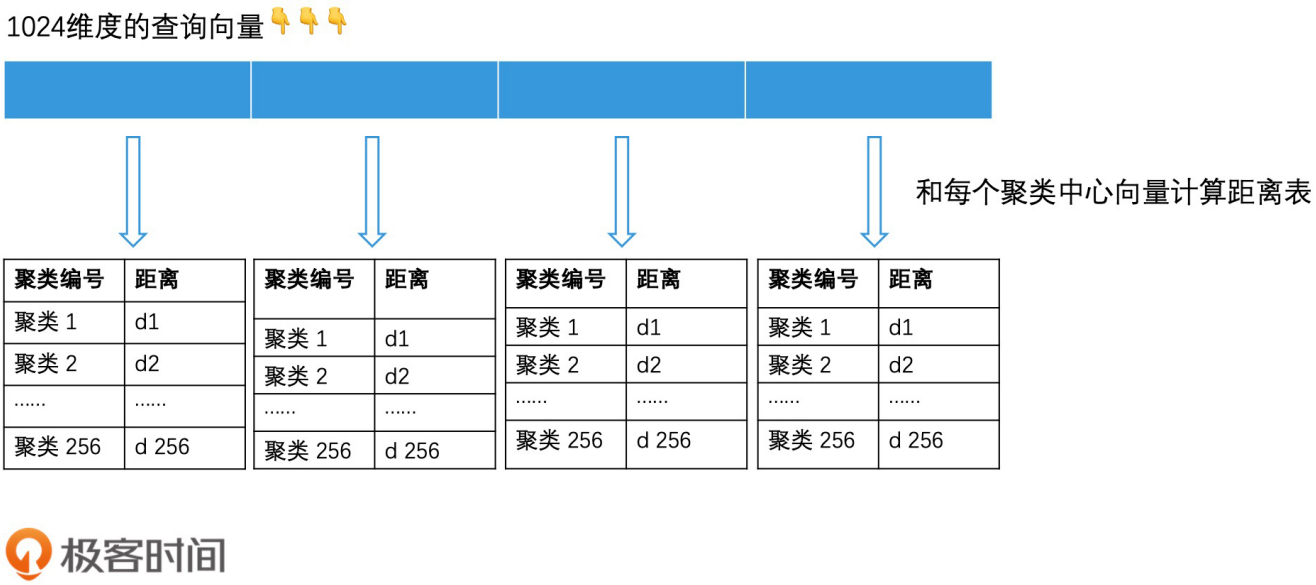
这样，我们就得到了一个压缩后的样本向量，它是一个 32 个比特位的向量。这个时候，如果要我们查询一个新向量和样本向量之间的距离，也就是它们之间的相似性，我们该怎么做呢？这里我要强调一下，一般来说，要查询的新向量都是一个未被压缩过的向量。也就是说在我们的例子中，它是一个 1024 维的浮点向量。

好了，明确了这一点之后，我们接着来说一下计算过程。这整个计算过程会涉及 3 个主要向量，分别是**样本向量**、**查询向量**以及**聚类中心向量**。你在理解这个过程的时候，要注意分清楚它们。

那接下来，我们一起来看一下具体的计算过程。

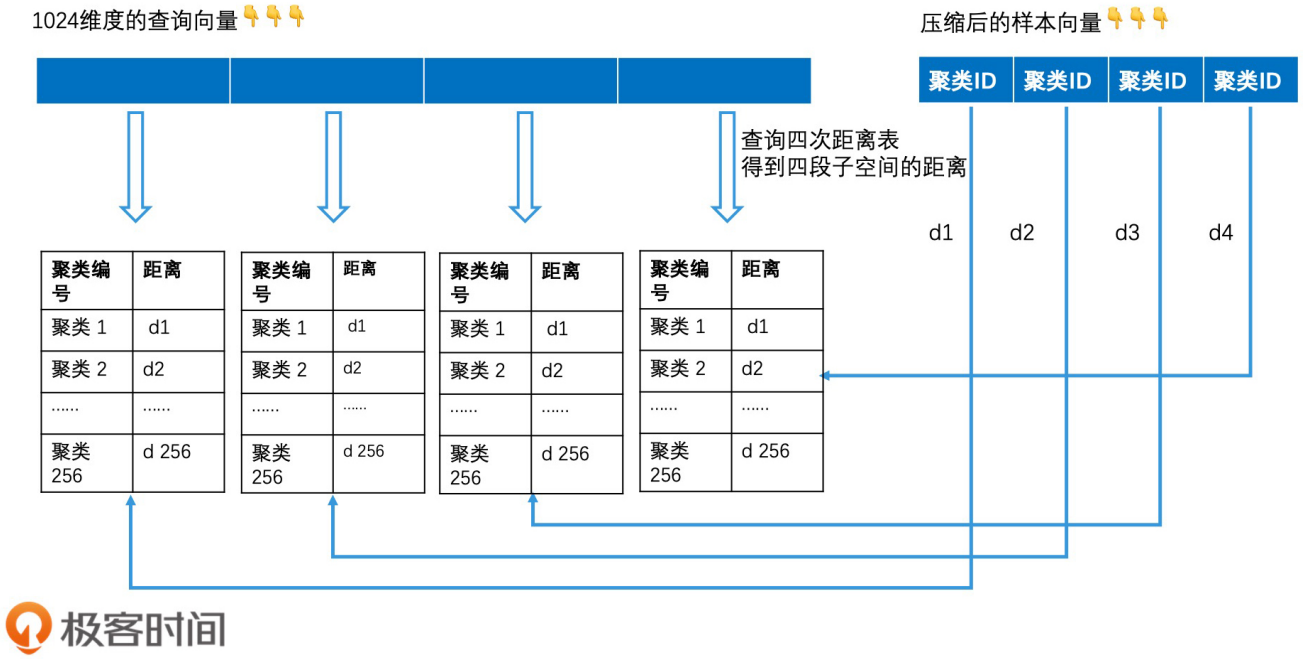
首先，我们在对所有样本点生成聚类时，需要记录下**聚类中心向量**的向量值，作为后面计算距离的依据。由于 1024 维向量会分成 4 段，每段有 256 个聚类。因此，我们共需要存储 1024 个聚类中所有中心向量的数据。

然后，对于**查询向量**，我们也将它分为 4 段，每段也是一个 256 维的向量。对于查询向量的每一段子向量，我们要分别计算它和之前存储的对应的 256 个聚类中心向量的距离，并用一张距离表存下来。由于有 4 段，因此一共有 4 个距离表。



计算查询向量和聚类中心向量的距离表过程示意图

当计算查询向量和样本向量的距离时，我们将查询向量和样本向量都分为 4 段子空间。然后分别计算出每段子空间中，查询子向量和样本子向量的距离。这时，我们可以用聚类中心向量代替样本子向量。这样，求查询子向量和样本子向量的距离，就转换为求查询子向量和对应的聚类中心向量的距离。那我们只需要将样本子向量的聚类 ID 作为 key 去查距离表，就能在 O(1) 的时间代价内知道这个距离了。



获得全部查询子向量和样本子向量近似距离的过程示意图

最后，我们将得到的四段距离按欧氏距离的方式计算，合并起来，即可得到查询向量和样本向量的距离，距离计算公式：

$$\text{距离} = \sqrt[2]{d1^2 + d2^2 + d3^2 + d4^2}$$

极客时间

以上，就是计算查询向量和样本向量之间距离的过程了。你会看到，原本两个高维向量的复杂的距离计算，被 4 次 O(1) 时间代价的查表操作代替之后，就变成了常数级的时间代价。因此，在对压缩后的样本向量进行相似查找的时候，我们即便是使用遍历的方式进行计算，时间代价也会减少许多。

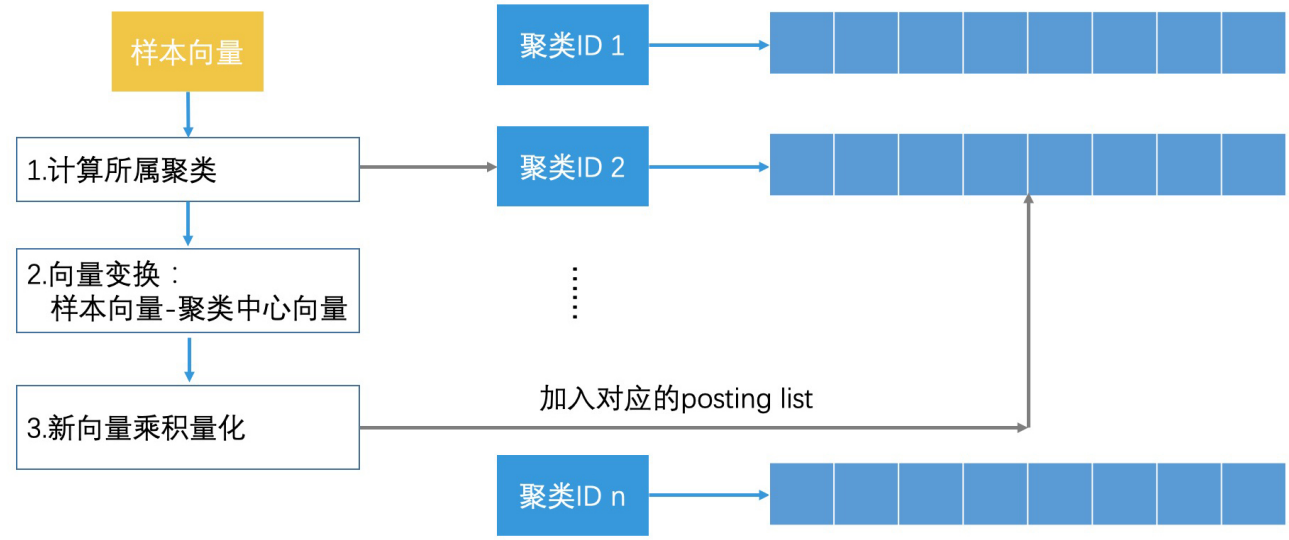
而计算查询向量到每个聚类中心的距离，我们也只需要在查询开始的时候计算一次，就可以生成 1024 个距离表，在后面对比每个样本向量时，这个对比表就可以反复使用了。

如何对乘积量化进行倒排索引？

尽管使用乘积量化的方案，我们已经可以用很低的代价来遍历所有的样本向量，计算每个样本向量和查询向量的距离了。但是我们依然希望能用更高效的检索技术代替遍历，来提高检索效率。因此，结合前面的知识，我们可以将聚类、乘积量化和倒排索引综合使用，让整体检索更高效。下面，我就来具体说说，在建立索引和查询这两个过程中，它们是怎么综合使用的。

首先，我们来说建立索引的过程，我把它总结为 3 步。

1. 使用 K-Means 聚类，将所有的样本向量分为 1024 个聚类，以聚类 ID 为 Key 建立倒排索引。
2. 对于每个聚类中的样本向量，计算它们和聚类中心的差值，得到新的向量。你也可以认为这是以聚类中心作为原点重新建立向量空间，然后更新该聚类中的每个样本向量。
3. 使用乘积量化的方式，压缩存储每个聚类中新的样本向量。

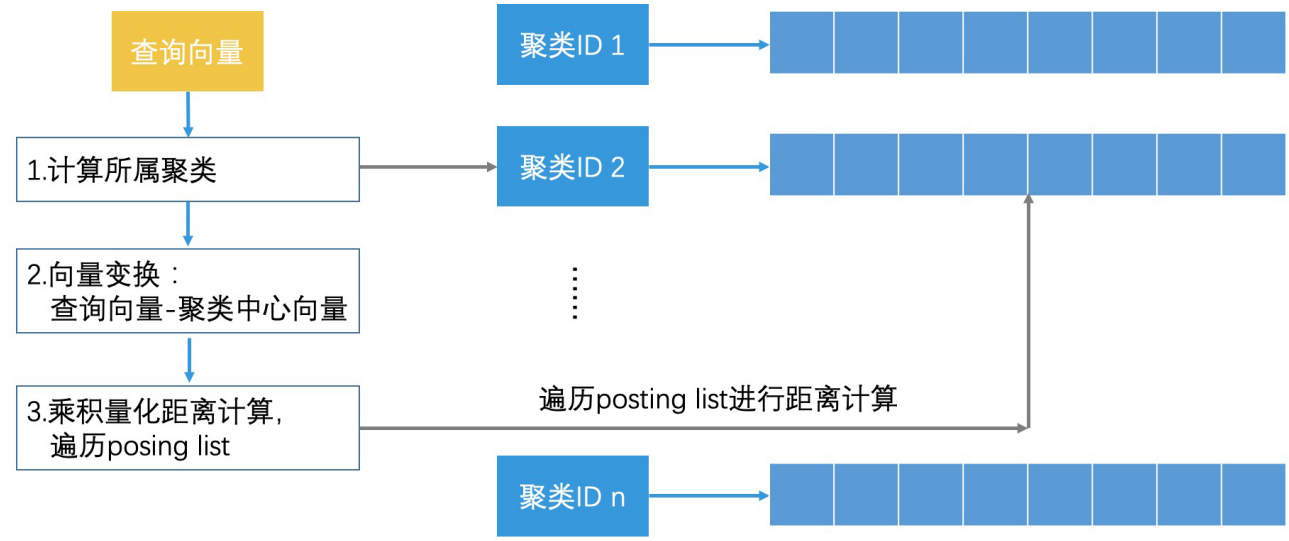


极客时间

一个样本向量加入倒排索引的过程示意图

建好索引之后，我们再来说说查询的过程，它也可以总结为 3 步。

1. 当查询向量到来时，先计算它离哪个聚类中心最近，然后查找倒排表，取出该聚类中所有的向量。
2. 计算查询向量和聚类中心的差值，得到新的查询向量。
3. 对新的查询向量，使用乘积量化的距离计算法，来遍历该聚类中的所有压缩样本向量，取出最近的 k 个结果返回。



极客时间

查询向量查询倒排索引的过程示意图

这样，我们就同时结合了聚类、乘积量化和倒排索引的检索技术，使得我们能在压缩向量节省存储空间的同时，也通过快速减少检索空间的方式，提高了检索效率。通过这样的组合技术，我们能解决大量的图片检索问题。比如说，以图搜图、拍照识物，人脸识别等等。

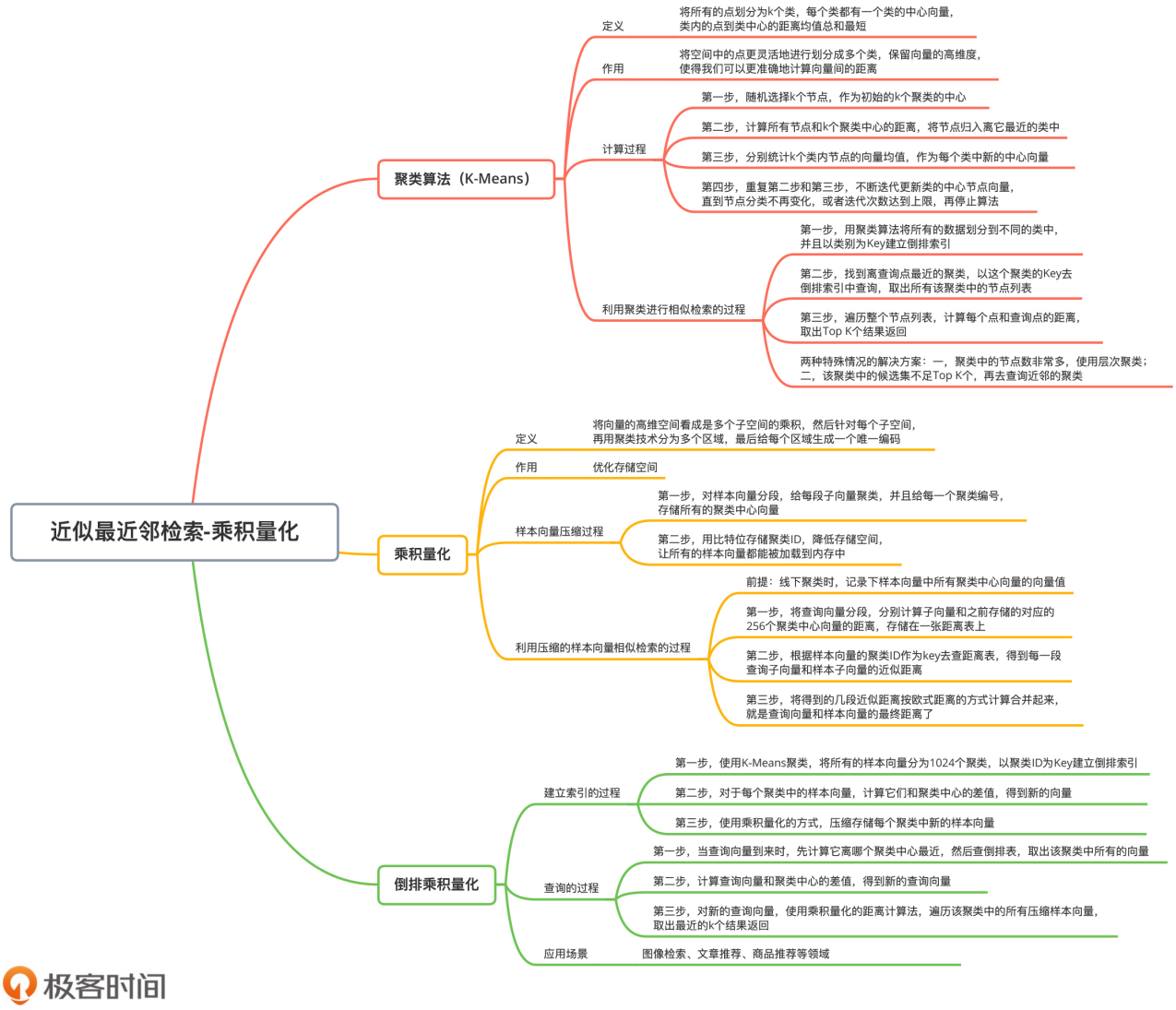
实际上，除了图像检索领域，在文章推荐、商品推荐等推荐领域中，我们也都可以用类似的检索技术，来快速返回大量的结果。尤其是随着 AI 技术的发展，越来越多的对象需要用特征向量来表示。所以，针对这些对象的检索问题，其实都会转换为高维空间的近似检索问题，那我们今天讲的内容就完全可以派上用场了。

重点回顾

今天，我们学习了在高维向量空间中实现近似最邻近检索的方法。相对于局部敏感哈希，使用聚类技术能实现更灵活的分类能力，并且聚类技术还支持层次聚类，它能更快速地划分检索空间。

此外，对于高维的向量检索，如何优化存储空间也是我们需要考虑的一个问题。这个时候，可以使用乘积量化的方法来压缩样本向量，让我们能在内存中运行向量检索的算法。

那为了进一步提高检索率和优化存储空间，我们还能将聚类技术、乘积量化和倒排索引技术结合使用。这也是目前图像检索和文章推荐等领域中，非常重要的设计思想和实现方案。



知识总结

课堂讨论

1. 为什么使用聚类中心向量来代替聚类中的样本向量，我们就可以达到节省存储空间的目的?

2. 如果二维空间中有 16 个点，它们是由 x 轴的 1、2、3、4 四个点，以及 y 轴的 1、2、3、4 四个点两两相乘组合成的。那么，对于二维空间中的这 16 个样本点，如果使用乘积量化的思路，你会怎么进行压缩存储？当我们新增了一个点 (17,17) 时，它的查询过程又是怎么样的?

欢迎在留言区畅所欲言，说出你的思考过程和最终答案。如果有收获，也欢迎把这一讲分享给你的朋友。

学习计划

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　15 | 最近邻检索（上）：如何用局部敏感哈希快速过滤相似文章?

下一篇　特别加餐 | 高性能检索系统中的设计漫谈

精选留言 (5)

写留言

王创峰
2020-05-03

对于乘积法省空间的说法有点疑问.老师在文章里的说法确实能看出节省了空间,但是这里面临的问题是:给定16个二维空间的点,使用(x,y)的存储方法,会比分别存储x和y的方法更省空间吗?假如这16个点的x值分布在16个值上,y值也分布在16个值上,那两种方法都需要32个浮点数表示啊?怎么就能省空间呢?

展开

作者回复: 首先，你举的这个例子不叫“乘积”，而只是将16个(x，y)分别存成16个(x)和16个(y)，这样的确是没有省空间的。
然后，我文中的点，其实代表的是“聚类中心向量”，而不是“样本向量”。接下来我来和你一起分析一下。
如果二维空间按格子划分区域，一共有16个区域，那么我们就需要16个编号(相当于生成16个聚

类，每个聚类要记录自己的中心向量，有16个二维的中心向量)。但如果我们把二维空间分解为x轴和y轴，那么x轴只需要4个编号，y轴只需要4个编号。这样我们只需要4+4个编号既可以表示完整的二维空间了(相当于将向量分为两个子空间，每个子空间生成4个聚类，每个聚类需要记录自己的中心向量，这样就有8个一维的中心向量)

那么，记录8个一维的中心向量比记录16个二维的中心向量省空间。空间是省在了这里，而不是省在样本向量上。

对于每个样本向量，之前的二维空间编号是1 - 16，需要4个比特位，分解后的两个一维空间编号是(1 - 4)*(1 - 4)，也是需要4个比特位。因此，样本向量的存储，无论是个数，还是压缩后需要用到比特位，都是无法节省的。

总结一下，乘积的好处，其实是可以降低我们要存储的中心向量的个数和维度，而不是降低样本向量的个数。如果样本向量原来有n个，那么乘积量化后依然有n个。我文中举的“点”的例子，结合后面的内容看，你就会发现它代表的是聚类中心向量，而不是样本向量。我想这可能就是让你觉得疑惑的地方，希望这个分析可以解答你的疑惑。

1

那时刻

2020-05-01

请问在聚类算法进行相似检索中，对于聚类中的候选集不足 Top K 个，我们还可以再去查询次邻近的聚类，这个临近聚类怎么来寻找呢？是重新计算当前聚类中心点和其它中心点距离么？还是需要额外存储聚类中心点之间的距离？

展开

作者回复: 是这样的，在k-means算法中，我们会生成k个聚类，每个聚类都有自己的聚类中心向量。这些中心向量会存储好。

然后，当查询向量到来的时候，我们会和这k个聚类中心向量比较，这样就能得到查询向量到每个聚类的距离。最近的那个聚类，就是目标聚类。如果这个目标聚类中的候选集不足，那么距离第二近的聚类，就是次临近的聚类。同理，如果次邻近的这个聚类中候选集还是不足，那么我们可以再去找距离第三近的聚类。依此类推。

那时刻

2020-05-01

非常感谢老师这一节内容，收获不少。对于高维度的向量进行乘积量化，颇类似于数据库里对于数据分表操作，分散存储和提高查询速度。

尝试回答一下讨论问题

1. 使用聚类中心向量来代替聚类中的样本向量，其一，这样不需要存储每个聚类里样本...

展开

作者回复: 今天的内容的确会比较难，知识点也比较多，所以总结部分有一个思维导图，希望能帮你更好地吸收。

关于思考题:

1. 压缩的根源，是在于我们可以用1个类的中心向量来代替类内所有的n个样本向量。用你熟悉的数据库分表来做例子，分析如下:

原来的向量存储是这样的:

样本向量表结构:[样本向量ID | 向量数据]

每个向量数据都是一个高维向量，会很占空间。

但用聚类中心向量代替样本向量以后，我们会有两张表:

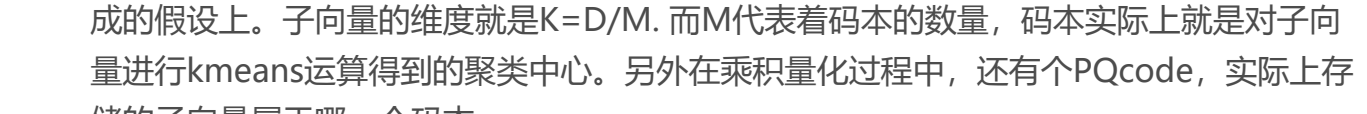
聚类中心表结构:[聚类ID | 聚类中心向量数据]

样本向量表结构:[样本向量ID | 聚类ID]

你会发现，和原先的相比，我们加了一个聚类中心表，由于聚类不多，因此这个表不会太大。而原来的样本向量表，它存的数据从“向量数据”变成了“聚类ID”，从而不用存储具体的向量(可以用聚类ID关联到聚类中心表)，因此表空间得到压缩。

2.是的，这题是希望你能更好地理解乘积量化的思想，并不一定要求要严格按照每个流程做。我们可以分为xy轴两个维度(这样就是乘积了)。不过由于数据太少，所以也不需要用聚类，比如你把x轴等分2段也可以。然后每段编上号(这样就是量化了)。

然后查询时，你分别和x轴和y轴比，找到对应的区域就好。



也许这就是因为门槛高了，才使得做算法和 AI 的同学工资高

想想从小学开始一直到大学(甚至有些读到了研究生和博士)，也都是从简单开始学起，最...
展开

作者回复: 实话实说，这一讲的内容的确很难。即使是我自己当年学这些内容的时候，也琢磨了好久。而且这一部分的内容和AI结合较紧密，也是随着这几年AI的大面积铺开才开始变得重要起来的，之前大部分工程师可能并不熟悉。因此，多看几遍，有不懂的地方一起讨论分析，相信你会有收获的。

为什么进阶篇最后一篇讲了这样一个内容呢，你会发现，我们的应用场景都在进化，现在搜索引擎已经支持图片搜索了，电商平台也支持拍照购物，还有人脸识别等应用，对应起来，我们的检索技术，已经需要从文本关键词检索，升级到图片检索，视频检索等更复杂的空间中了。

包括现在的广告系统，其实各种智能匹配，都可以看着是向量匹配，而不仅仅是传统的关键词倒排检索。

我希望我们都能跟着时代的潮流，提升自己的认知和竞争力。

由于专栏篇幅有限，因此多少会有一些知识没有展开介绍。不过，你可以把这个专栏的内容当做一个索引，然后对于自己感兴趣的部分去深入学习了解。有不清楚的地方，也欢迎讨论和交流。